‘

Using the Yale HPC Clusters

Robert Bjornson

Yale Center for Research Computing
Yale University

Mar 5, 2019

Getting the Workshop Slides and Demos

- https://github.com/ycrc/Intro-Bootcamp
- download zip and unpack
- or: git clone https://github.com/ycrc/Intro-Bootcamp.git
- slides are hpc.pdf

What is the Yale Center for Research Computing?

- Independent center under the Provost's office
- Created to support your research computing needs
- Focus is on high performance computing and storage
- ˜15 staff, including applications specialists and system engineers
- Available to consult with and educate users
- Manage compute clusters and support users
- Located at 160 St. Ronan st, at the corner of Edwards and St. Ronan
- http://research.computing.yale.edu
- http://docs.ycrc.yale.edu

A note on documentation

Rather than provide urls to our documentation, I'll use:

`DOCS:clusters`

Which means: at docs.ycrc.yale.edu, search for "clusters"

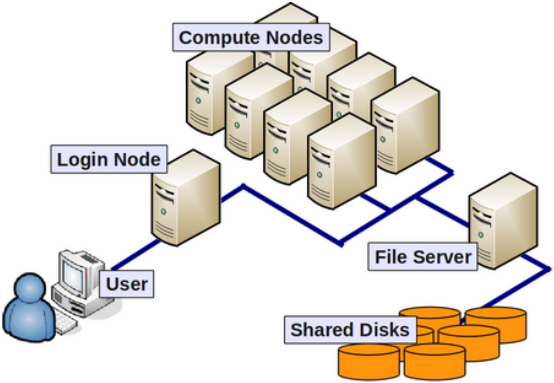
What is a cluster?

A **cluster** usually consists of a hundred to a thousand rack mounted computers, called **nodes**. It has one or two **login nodes** that are externally accessible, but most of the nodes are compute nodes and are only accessed from a login node via a **batch queueing system** (Slurm).

The CPU used in clusters may be similar to the CPU in your desktop computer, but in other respects they are rather different.

- Linux operating system
- Command line oriented
- Many cores (cpus) per node
- No monitors, no CD/DVD drives, no audio or video cards
- Lots of RAM
- Very large distributed file system(s)
- Connected internally by a fast network

Compute Cluster Schematic



Yale Compute Cluster

Why use a cluster?

Clusters are very powerful and useful, but it may take some time to get used to them. Here are some reasons to go to that effort:

- Don't want to tie up your own machine for many hours or days
- Have many long running jobs to run
- Want to run in parallel to get results quicker
- Need more disk space
- Need more memory
- Want to use software installed on the cluster
- Want to access data stored on the cluster
- Want to use GPUs
- Cluster use is free (within reason)
- YCRC user support

Limitations of clusters

Clusters are not the answer to all large scale computing problems. Some of the limitations of clusters are:

- Cannot run Windows or Mac programs
- Not for persistent services (DBs or web servers)
- Not ideal for graphical tasks
- Jobs that run for weeks can be a problem (unless checkpointed)

Summary of Yale Clusters (Mar 2019)

Cluster	Users	Role	CPUs
Omega	FAS	Highly Parallel/MPI	6,500
Grace	FAS	General Purpose	10,000
Farnam	YSM/LS	General Purpose	5,500
Ruddle	YCGA	Sequence Data	3,000
Milgram	Psychology	HIPAA Data	1,600

Details on each cluster at DOCS:clusters

Setting up a account

Accounts are free of charge to Yale researchers.

Request an account at:
`http://research.computing.yale.edu/account-request`.

After your account has been approved and created, you will receive an email describing how to access your account. This will involve setting up ssh keys Details vary by cluster.

If you need help setting up or using ssh, send an email to: `hpc@yale.edu`.

Ssh to a login node

To access any of the clusters, you must use **ssh**. From a Mac or Linux machine, you simply use the **ssh** command:

```
laptop$ ssh netid@{omega,grace,etc}.hpc.yale.edu
```

From a Windows machine we recommend using mobaXterm
`https://mobaxterm.mobatek.net`
or putty: `http://www.putty.org`.

For more information on mobaxterm, see DOCS: mobaxterm

Understanding ssh key pairs

- We use key pairs instead of passwords to log into clusters
- Keys are generated as pair:
    - ssh-keygen: public (id_rsa.pub) and private (id_rsa)
    - putty/mobaxterm/winscp: public (name.pub) and private (name.ppk)
- You should always use a pass phrase to protect private key!
- You can freely give the public key to anyone
- You can generate a new pair for each of your computers, or reuse the same pair.
- NEVER give the private key to anyone!

Outline of setting up keys

1. Generate key pair using ssh-keygen (linux/mac) or puttygen (windows)
2. Locate the public and private key files you generated
3. Use our tool to upload the public key (link on page shown below)
4. Wait 15 minutes for key to propagate
5. Connect using private key

This can be tricky the first time. See DOCS: ssh for the exact steps or contact us if you have problems.

Sshing to Ruddle

- Ruddle has an additional level of ssh security, using Multi Factor Authentication (MFA)
- We use Duo, the same MFA as other secure Yale sites

Example:

```
bjornson@debian:~$ ssh rdb9@ruddle.hpc.yale.edu
Enter passphrase for key '/home/bjornson/.ssh/id_rsa':
Duo two-factor login for rdb9

Enter a passcode or select one of the following options:

 1. Duo Push to XXX-XXX-9022
 2. Phone call to XXX-XXX-9022
 3. SMS passcodes to XXX-XXX-9022

Passcode or option (1-3): 1
Success. Logging you in...
```

More about MFA

- Don't have a smartphone? Get a hardware device from the ITS Helpdesk
- Register another phone: e.g your home or office phone, as a backup
- See DOCS: mfa

Cluster Storage

Each cluster has a number of different storage areas for you to use, with different rules and performance characteristics.

It is very important to understand the differences. Generally, each cluster will have:

Home : Backed up, small quota, for scripts, programs, documents, etc.

Scratch60 : Not backed up. Purged after 60 days. For temp files.

Project : Not backed up. For longer term storage.

PI : Not backed up. Some groups have dedicated storage.

Local HD : /tmp For local scratch files.

RAMDISK : /dev/shm For local scratch files.

Storage@Yale : University-wide storage (active and archive).

For more info, DOCS: Monitor CPU

Quotas

- All storage areas have quota limits, both size and file count
- Most limits are per group, exception: home on Ruddle and Farnam
- If you hit your limit, you won't be able to write files, so jobs fail
- To check your limit: `$ groupquota`
- Project and scratch60 have very large limits, home is relatively small.

Running jobs on a cluster

Two ways to run jobs on a cluster:
Interactive:

1. you request an allocation
2. system grants you one or more nodes
3. you are logged onto one of those nodes
4. you run commands
5. you exit and system automatically releases nodes

Batch:

1. you write a job script containing commands
2. you submit the script
3. system grants you one or more nodes
4. your script is automatically run on one of the nodes
5. your script terminates and system releases nodes
6. system sends a notification via email

Interactive vs. Batch

Interactive jobs:

- like a remote session
- require an active connection
- for development, debugging, or interactive environments like R and Matlab
- one or a few jobs at a time

Batch jobs:

- non-interactive
- can run many jobs simultaneously
- your best choice for production computing

General overview of Slurm

Slurm manages all the details of cluster operation:

- Interactive node allocation
- Batch job submission
- Specifying and reserving the resources you need for your job
- Listing running and pending jobs
- Cancelling jobs
- Grouping node resources into partitions
- Prioritizing and scheduling jobs

Slurm documentation: https://slurm.schedmd.com/

Interactive allocations

```
srun -p interactive --pty bash
```

You'll be logged into a compute node and can run your commands.

To exit, type exit or ctrl-d

Slurm: Example of an interactive job

```
farnam-0:~ $ srun --pty -p interactive --mem=8g bash
c01n01$ module load BWA
c01n01$ module load SAMtools
c01n01$ bwa mem BWAIndex/genome.fa Data/reads01.fa > reads01.sam
c01n01$ exit
farnam-0:~ $
```

Batch jobs

- you create a script wrapping your job
- declares resources required
- contains the command(s) to run

Slurm: Example of a batch script

```
#!/bin/bash
#SBATCH --mail-type=ALL
#SBATCH -t 3:00 # 3 minutes
#SBATCH --mem=10g

module load R

Rscript myscript.R
```

Slurm: Example of a batch job

```
$ sbatch batch.sh
Submitted batch job 42
$ squeue -j 42
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
   42   general   batch     rdb9  R       0:03      1 c13n10
```

The script runs in the current directory. Output goes to slurm-*jobid*.out by default, or use -o

Copy scripts and data to the cluster

- On Linux and Mac, use scp and rsync to copy scripts and data files from between local computer and cluster.

```
laptop$ scp -r ~/workdir netid@omega.hpc.yale.edu:
laptop$ rsync -av -e 'ssh -i ~/.ssh/id_rsa' ~/workdir \
        netid@grace.hpc.yale.edu:
```

- For Windows, you can use Mobaxterm for ssh, GUIs, file transfer
- Cyberduck is a good graphical tool for Mac or Windows. https://cyberduck.io
- Graphical copy tools often need configuration when used with Duo (Ruddle).
- DOC: transfer

Y | CRC

Summary of Slurm commands

Description	**Command**
Submit a batch job	sbatch [*opts*] *SCRIPT*
Submit an interactive job	srun -p interactive –pty [*opts*] bash
Status of a job	squeue -j *JOBID*
Status of a user's jobs	squeue -u *NETID*
Get job stats	seff *JOBID*
Cancel a job	scancel *JOBID*
Get job info for partition	squeue -p *PART*
Get node info for partition	sinfo -p PART
Get individual node info	sinfo -N -n *NODE*
Get job info	sacct -j *JOBID* -l — less -S

Useful formatting:

```
export SACCT_FORMAT="JobID%-20,JobName,User,Partition,NodeList,Elapsed,\
  State,ExitCode,MaxRSS, AllocTRES%32"
```

```
export SQUEUE_FORMAT="%.16i %.12P %.12j %.8u %.2t %.12M %.12l %20R %.6D \
  %.6C %.10m %8b %8f"
```

Summary of Slurm sbatch/srun options

Description	**Option**
Partition	-p *PART*
Process count	-c *CPUS*
Node count	-N *NODES*
Task count	-n *TASKS*
Wall clock limit	-t *HH*:*MM* or *DD-HH*
Memory limit	--mem 4g or --mem-per-cpu 4g)
Interactive job	srun --pty bash
Job name	-J *NAME*
Node constraint	-C *type*

Examples:

```
farnam1$ srun --pty -p general --mem-per-cpu 32g -t 24:00 bash
farnam1$ sbatch -p general --mem 8g -c 20 -t 3- job.sh
```

Partitions

- Compute nodes are grouped into partitions
- Each cluster has its own set of partitions: `sinfo -s` or webpage.
- Jobs are submitted to a specific partition
- Each partition has rules:
  - Who can submit jobs
  - How many cores and how much memory per user
  - Maximum walltime

interactive	for interactive jobs (srun)
general	default on farnam/ruddle
day	default on grace/omega
week	long jobs on grace/omega
gpu	nodes with gpus
bigmem	nodes with large RAM
*pi_**	reserved for specific groups
scavenge	uses idle nodes from other partitions (preempted)

Y | CRC

Partitions cont.

- scavenge can be very useful for short jobs
- You can submit to multiple partitions: `sbatch -p general,pi_zhao,scavenge`.

Nodes -N, Tasks -n, Cores -c

- The sbatch/srun options -N, -n, -c often cause confusion. They change the size of your allocation and its placement.
- By default, you get 1 task, having 1 core, running on 1 node
- -n *n* specifies n tasks. This is normally only useful with MPI.
- -c *c* causes each task to have c cpus/cores (on a node). This is very useful for multithreaded programs
- -N *N* forces job tasks to be scheduled onto exactly N nodes. Rarely useful.

Examples:

- `sbatch -c 20`: a single 20-core task on 1 node
- `sbatch -c 20 -n 4`: 4 20-core tasks, on multiple nodes
- `sbatch -n 40` : 40 1-core tasks, spread indeterminately among nodes

Controlling memory usage

- It's crucial to understand memory required by your program.
- Nodes (and thus memory) are often shared
- Jobs have modest default memory limits (5GB/core) that you can override
- Jobs exceeding their request are killed. Common errors: "bus error", or "Exceeded step memory at some point"
- `--mem=8g` requests 8GB per node
- `--mem-per-cpu=8g` requests 8GB per cpu. Multiplied by cores (-c)

Controlling memory usage (cont)

To determine how much memory your job uses:

- `seff <jobid>`
- `sacct -j <jobid> -l | less -S` (MaxRSS)
- /usr/bin/time -a *cmd* ... (maxresident)

To specify 8 cores on one node with 8 GB RAM for each core:

```
sbatch -c 8 --mem-per-cpu=8G t.sh
```

Controlling walltime

- Each job is assigned a maximum walltime
- If not specified, it will get the default walltime limit
- The job is killed if time is exceeded
- You can specify longer walltime to avoid the job being killed
- You can specify shorter walltime to get resources faster

To specify walltime limit of 2 days:

```
sbatch -t 2- t.sh
```

Modules

Software is setup with *module*

`$ module avail`	*find modules*
`$ module avail name`	*find particular module*
`$ module load name`	*use a module*
`$ module list`	*show loaded modules*
`$ module unload name`	*unload a module*
`$ module purge`	*unload all*
`$ module save collection`	*save loaded modules as collection*
`$ module restore collection`	*restore collection*
`$ module describe collection`	*list modules in collection*

Example “module load” commands

```
module load Python/3.5.1-foss-2016b
module load Perl
module load Bowtie2/2.2.9-foss-2016a
```

You can ask for the default or specify a specific version:

```
module load Python
module load Python/3.5.1-foss-2016b
```

Note that on grace and omega, module names are prefixed with a class:

```
module load Apps/R/2.15.3
```

Installing Software

What if software you need is not currently installed as a module?

- General interest, ask us to install it (hpc@yale.edu)
- R and Python packages, we've moved to DYI
    - Python: miniconda
    - R packages: install.packages
- bleeding-edge C/C++ apps: try DYI
- More info on our website
- We'll help if you run into problems

Interactive tools (Python/Matlab/R/Mathematica) as batch jobs

```
compute-20-1$ python compute.py input.dat
compute-20-1$ R --slave -f compute.R --args input.dat
compute-20-1$ matlab -nodisplay -nosplash -nojvm < compute.m
compute-20-1$ math -script compute.m
compute-20-1$ MathematicaScript -script compute.m input.dat
```

You often can get help from the command itself using:

```
compute-20-1$ matlab -help
compute-20-1$ python -h
compute-20-1$ R --help
```

Running graphical programs on compute nodes

Two different ways:

- X11 forwarding
  - easy setup
  - `ssh -Y` to cluster, then `srun --x11`
  - works fine for most applications
  - bogs down for very rich graphics
- Remote desktop (VNC)
  - more setup
  - allocate node, start VNC server there, connect via ssh tunnels
  - works very well for rich graphics
- Coming soon to Farnam: Open On Demand, web-based desktop

More information, DOC: X11, VNC

Use Pipes to improve performance

Faster: avoids file IO and increases parallelism

Using Files

```
$ gunzip test_export.txt.gz
$ perl filter.pl test_export.txt \
  test_filtered.txt
$ perl illumina_export2sam.pl \
  --read1=test_filtered.txt > test_filtered.sam
$ samtools view -bS -t hg19.fa.fai \
  test_filtered.sam -o test_filtered.bam
$ samtools sort test_filtered.bam test_sorted
```

Using Pipes

```
$ gunzip -c test.export.txt.gz \
| perl filter.pl - - \
| perl illumina\_export2sam.pl --read1=- \
| samtools view -bS -t hg19.fa.fai - \
| samtools sort - test.sorted
```

Large datasets (e.g. Genomes)

- Please do not install your own copies of popular files (e.g. genome refs).
- We have a number of references installed, and can install others.
- For example, on Ruddle: /home/bioinfo/genomes
- If you don't find what you need, please ask us, and we will install them.

Issues with CPU architectures

- Our clusters are heterogeneous, with multiple cpu types
- Code compiled on new nodes (with avx2, avx512) often fails on older nodes (avx)
- Typical error: "Instruction Error"
- Solutions:
  1. Always compile on the oldest node type (we do this).
  2. Always run on the newest node type
- Consult page for your cluster and use -C to choose node type.
- DOC: illegal instruction

Wait, where is the Parallelism?

Sbatch can allocate multiple cores and nodes, but the script runs on one core on one node sequentially.

Simply allocating more nodes or cores DOES NOT make jobs faster.

How do we use multiple cores to increase speed?

Two classes of parallelism:

- Single job parallelized (somehow)
- Lots of independent sequential jobs

Some options:

- Submit many batch jobs simultaneously (not good)
- Use job arrays, or dSQ (much better)
- Submit a parallel version of your program (great if you have one)

Job Arrays

- Useful when you have many nearly identical, independent jobs to run
- Starts many copies of your script, distinguished by a task id.

Submit jobs like this:

```
Slurm: sbatch --array=1-100 ..
```

Your batch script uses an environment variable:

```
./mycommand -i input.${SLURM_ARRAY_TASK_ID} \
    -o output.${SLURM_ARRAY_TASK_ID}
```

dSQ (aka Dead Simple Queue)

- Useful when you have many similar, independent jobs to run
- Automatically schedules jobs onto a single PBS allocation

Advantages

- Handles startup, shutdown, errors
- Only one batch job to keep track of
- Keeps track of status of individual jobs
- More flexible than job arrays
- Automatically schedules jobs onto a single Slurm allocation
- Limits (walltime, memory, cpus, etc) are *per individual job*

Using dSQ

1. Create file containing list of commands to run (jobs.txt)

   ```
   prog arg1 arg2 -o job1.out
   prog arg1 arg2 -o job2.out
   ...
   ```

2. Create launch script

   ```
   module load dSQ
   dSQ --taskfile jobs.txt [slurm args] > run.sh
   ```

3. Submit launch script

   ```
   sbatch run.sh
   ```

For more info, DOC: dSQ

Parallel-enabled programs

- Many modern programs are able to use multiple cpus on one node.
- Typically specify something like `-p 20` or `-t 20` (see man page)
- You must allocate matching number of cores: -c 20

```
#!/bin/bash
#SBATCH -c 20

myapp -t 20 ...
```

MPI-enabled programs

- Some programs use MPI to run on many nodes
- Impressive speedups are possible
- MPI programs are compiled and run under MPI
- MPI must cooperate with Slurm

```
#!/bin/bash
#SBATCH -N 4 -n 20 --mem-per-cpu 4G

module load MPI/OpenMPI
mpirun ./mpiprog ...
```

GPU-enabled programs

Many programs are now enabled for GPUs: imaging, machine learning...
To use them, your program must be explicitly gpu enabled and compiled. Your slurm job must request a partition with gpu nodes and explicitly request gpus:

```
srun --pty -p gpu -c 20 --gres=gpu:2 bash
```

We have a number of generations of gpus: k80, p100, 1080ti, depending on cluster.
For more info, DOC: gpus and cuda

Best Practices

- Start Slowly
  - Run your program on a small dataset interactively
  - In another ssh, watch program with top. Track memory usage.
  - Check outputs
  - Only then, scale up dataset, convert to batch run
- Input/Output
  - Think about input and output files, number and size
  - Should you use local /tmp or /dev/shm?
- Memory
  - Use sacct, top or /usr/bin/time -a to monitor usage
  - Specify memory and walltime requirements
- Learn Linux! Almost all HPC (world-wide) is linux-based.
- Be considerate! Clusters are shared resources.
  - Don't run programs on the login nodes. Use a compute node.
  - Don't submit a huge number of jobs. Use dSQ or job array.
  - Don't do heavy IO to /home.
  - Keep an eye on your storage quotas!

Plug for scripting languages

- Learning basics of a scripting language is a great investment.
- Very useful for automating lots of day to day activities
  - Parsing data files
  - Converting file formats
  - Verifying collections of files
  - Creating processing pipelines
  - Summarizing, etc. etc.
- Python (strongly recommended)
- Bash
- Perl (if your lab uses it)
- R (if you do a lot of statistics or graphing)

To get help

- Send an email to: `hpc@yale.edu`
- Email me: `robert.bjornson@yale.edu`
- Read documentation at: `http://docs.ycrc.yale.edu`
- Come to office hours: `http://research.computing.yale.edu/support/office-hours-getting-help`

Resources

- This presentation: https://github.com/ycrc/Intro-Bootcamp
- Our other courses: Python, R, Git, Linux, etc: http://research.computing.yale.edu/training
- Table of equivalent Batch Queueing commands: https://slurm.schedmd.com/rosetta.pdf
- Linux: http://www.ee.surrey.ac.uk/Teaching/Unix or http://ryanstutorials.net
- Slurm: http://slurm.schedmd.com